دانشگاه اصفهان
دانشکده مهندسی کامپیوتر

مستند پروژه اول (رگرسیون خطی چند متغیره)

درس مبانی و کاربردهای هوش مصنوعی

دکتر کارشناس

اعضای گروه:

متین اعظمی (۴۰۰۳۶۲۳۰۰۳)

امیرعلی لطفی (۴۰۰۳۶۱۳۰۵۳)

زهرا معصومی (۴۰۰۳۶۲۳۰۳۴)

پاییز ۱۴۰۲

خواندن و آماده‌سازی داده‌ها

در ابتدا فایل csv با دستور زیر خوانده می‌شود:

```
df = pd.read_csv('Flight_Price_Dataset_Q2.csv')
```

سپس مقادیر مختلف هر ستون بررسی و بازبینی می‌شود.

شرح ستون‌های data frame مورد استفاده به صورت زیر خواهد بود:

- Stops: تعداد توقف‌های پرواز که دارای 3 مقدار zero و one و two_or_more می‌باشد.
- Departure_time: زمان شروع پرواز که دارای 6 مقدار زیر می‌باشد:
  - Early_Morning
  - Morning
  - Afternoon
  - Evening
  - Night
  - Late_Night
- Arrival_time: زمان پایان پرواز و رسیدن به مقصد که مقادیر آن مشابه با زمان شروع پرواز است.
- Duration: مدت زمان پرواز
- Days_left: روزهای باقی‌مانده تا پرواز
- Price: هزینه سفر
- Class: کلاس پرواز که مقدار آن می‌تواند Economy یا Business باشد.

تبدیل ستون‌های دسته‌ای به مقادیر عددی

1. **ستون Class:** با توجه به این که این ستون یک ویژگی اسمی است و ترتیب در آن مهم نیست، با استفاده از روش کدگذاری One_Hot آن را به مقدار عددی تبدیل می‌کنیم:

```
df = pd.get_dummies(df, columns=['class'], drop_first=True)
```

در اثر این عمل، دو ستون class_Economy و class_Business به دیتافریم اضافه می‌شود که کلاس پرواز را در هر ستون با True و False مشخص می‌کند. با توجه به این که این ویژگی کلاً دو مقدار Economy یا Business را شامل است (و هر ردیفی که Economy نیست قطعاً Business است و برعکس)، ستون class_Business را حذف کرده و تنها ستون

class_Economy را برای مشخص کردن کلاس پرواز استفاده می‌کنیم.

2. **ستون‌های arrival_time و departure_time:** این ستون‌ها، شامل ویژگی‌های ترتیبی هستند؛ پس نیاز داریم آن‌ها را به روشی کدگذاری کنیم که ترتیب آن‌ها حفظ شود. برای این کار از روش کدگذاری برچسبی (**Label Encoding**) استفاده می‌کنیم:

```
mapping = {'Early_Morning': 0,
           'Morning' : 1,
           'Afternoon' : 2,
           'Evening' : 3,
           'Night': 4,
           'Late_Night': 5}

df['arrival_time_encode'] = df['arrival_time'].map(mapping)
df['departure_time_encode'] = df['departure_time'].map(mapping)
df = df.drop('arrival_time', axis = 1)
df = df.drop('departure_time', axis = 1)
```

مقادیر Early_Morning تا Late_Night را به ترتیب به اعداد 0 تا 5 و بعد ستون‌های مربوط به زمان شروع و پایان پرواز را به ستون‌های کدگذاری معادل آن‌ها نگاشت می‌کنیم. سپس ستون‌های قبلی (با مقادیر غیر عددی) که دیگر به آن‌ها نیاز نداریم را از دیتافریم حذف می‌کنیم.

3. **ستون Stops:** این ستون نیز مقادیر ترتیبی دارد، پس با استفاده از یک مپ مقادیر صفر، یک و دو یا بیشتر را به اعداد 0 تا 2 نگاشت می‌کنیم:

```
mapping = {'zero': 0,
           'one' : 1,
           'two_or_more' : 2}

df['stops_encode'] = df['stops'].map(mapping)
df = df.drop('stops', axis = 1)
```

حالا دیتافریم ما به این صورت در می‌آید:

	duration	days_left	price	class_Economy	arrival_time_encode	departure_time_encode	stops_encode
0	2.17	1	5953	True	4	3	0
1	2.33	1	5953	True	1	0	0
2	2.17	1	5956	True	0	0	0
3	2.25	1	5955	True	2	1	0
4	2.33	1	5955	True	1	1	0
...	...	...	...	...	...	...	...
270133	17.25	49	68739	False	4	0	1
270134	10.08	49	69265	False	3	1	1
270135	10.42	49	77105	False	4	2	1
270136	10.00	49	81585	False	3	0	1
270137	10.08	49	81585	False	3	1	1

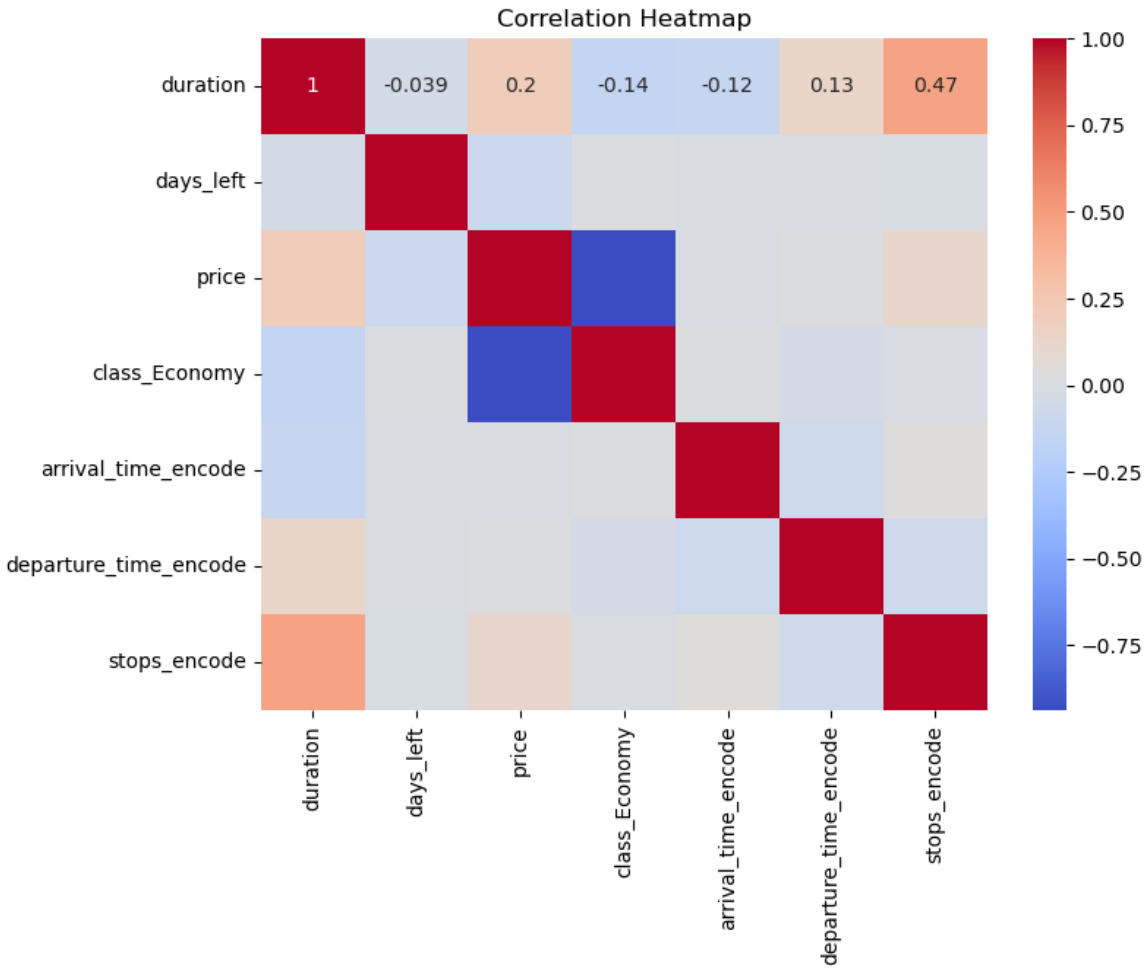
270138 rows × 7 columns

همبستگی ویژگی‌ها

برای بررسی میزان همبستگی ویژگی‌ها، از کتابخانه‌های matplotlib و seaborn برای نمایش HeatMap آن استفاده می‌کنیم:

```
plt.figure(figsize=(8,6))
sns.heatmap(df.corr(), annot = True, cmap = 'coolwarm')
plt.title("Correlation Heatmap")
plt.show()
```

با از استفاده از توابع بالا و `df.corr()` برای به‌دست آوردن همبستگی، نتیجه اجرا به صورت زیر خواهد بود:



این نقشه به ما نشان می‌دهد:

1. وابستگی هزینه پرواز و کلاس آن به هم بسیار زیاد و در جهت برعکس است. یعنی در صورتی که ویژگی class_Economy، مقدار True (یا 1) داشته باشد هزینه کاهش و در صورتی که False (یا 0) باشد هزینه افزایش خواهد یافت. (همبستگی -1)
2. زمان پرواز و تعداد توقف‌ها رابطه نزدیک و مستقیم با هم دارند؛ با افزایش تعداد توقف‌ها زمان پرواز بیشتر خواهد شد. (همبستگی 0.47)
3. هزینه پرواز و زمان آن رابطه نسبتاً نزدیک و مستقیم با هم دارند؛ با افزایش زمان پرواز، هزینه آن نیز افزایش خواهد یافت. (همبستگی 0.2)

وابستگی سایر متغیرها به هم ناچیز و قابل چشم‌پوشی است و اطلاعات خاصی را در اختیار ما قرار نمی‌هد.

پس از بررسی ویژگی‌های کلی دیتاست، به آماده‌سازی داده‌ها برای رگرسیون می‌پردازیم.

جدا کردن متغیرهای مستقل (ورودی) و وابسته (خروجی)

ابتدا مجموعه داده‌های X_df به عنوان داده‌های مستقل (ستون های زمان، تعداد توقف و ... که در تعیین قیمت پرواز اثرگذار هستند) و y_df را که همان ستون قیمت پرواز است را به عنوان متغیر وابسته از دیتافریم جدا می‌کنیم:

```
X_df = df.drop(columns=['price'])
y_df = df['price']
```

همچنین برای انجام محاسبات عددی روی داده‌ها، مقادیر True و False موجود در ستون کلاس پرواز را تبدیل به 0 و 1 می‌کنیم:

```
X_df['class_Economy'] = X_df['class_Economy'].astype(int)
```

جدا کردن مجموعه داده‌های آموزش و آزمایش

ما در این برنامه، از 80% داده‌ها برای آموزش (train) مدل و از 20% باقی‌مانده برای آزمایش (test) خروجی و ارزیابی نتیجه مدل استفاده می‌کنیم:

```
x_train, x_test , y_train , y_test =
train_test_split(X_df, y_df, test_size=0.2, random_state=4,
shuffle=True)
```

سپس برای راحتی کار و انجام عملیات برداری، لیست‌ها را به آرایه‌های numpy و y را به بردار ستونی تبدیل می‌کنیم:

```
X = x_train.to_numpy()
y = y_train.to_numpy()
y = y.reshape(-1, 1)
```

نرمال کردن داده‌ها (Normalization)

برای دریافت نتیجه دقیق از مدل در حال آموزش، نیاز داریم داده‌های نرمال‌شده را برای آموزش فراهم کنیم. برای این کار، مجموعه داده‌های آموزش و آزمایش را با استفاده از میانگین و انحراف معیار داده‌ها در محدوده مشخص قرار می‌دهیم:

```
X = (X - np.mean(X, axis=0)) / np.std(X, axis=0)
x_test = (x_test - np.mean(x_test, axis=0)) / np.std(x_test, axis=0)
```

پس از آماده‌سازی نهایی داده‌ها و متغیرها، به پیاده‌سازی تابع رگرسیون خطی با استفاده از گرادیان کاهشی می‌پردازیم.

پیاده‌سازی رگرسیون خطی چند متغیره

برای پیاده‌سازی این بخش، ما از تعاریف زیر استفاده خواهیم کرد:

- **نمونه‌های آموزشی (X):** ستون‌هایی که از دیتافریم جدا کردیم تا برای آموزش مدل استفاده کنیم.
- **ضرایب ویژگی‌ها (W):** میزان تاثیر هر یک از ویژگی‌ها در تعیین قیمت پرواز را مشخص می‌کند. این ضرایب به صورت یک بردار 6 تایی (چون 6 ویژگی داریم) تعریف شده است.
- **عرض از مبدا (b):** مقدار ثابتی به عنوان bias تابع.
- **مقادیر پیش‌بینی شده (y_hat):** مقادیری که مدل با استفاده از داده‌های آموزشی برای قیمت پرواز پیش‌بینی می‌کند.
- **مقادیر واقعی (y):** مقدار واقعی قیمت پرواز که از دیتافریم جدا کردیم.

در رگرسیون خطی، مجموعه‌ای از ورودی‌ها (features) و خروجی‌ها (targets) را به یک مدل (f) می‌دهیم که با استفاده از آن‌ها، عملکردی (یک تابع) برای پیش‌بینی خروجی برای ورودی‌های جدید تعریف می‌شود. به مقدار پیش‌بینی شده $\hat{y}$ می‌گوییم. این مدل به صورت زیر تعریف می‌شود:

$$f_{w,b}(X) = X^T W + b$$

که در آن X بردار ویژگی‌ها، W بردار ضرایب و b پارامتری برای تعیین مقدار اولیه و عرض از مبدا است.

تابع هزینه

برای ارزیابی عملکرد مدل، از تابع هزینه استفاده می‌کنیم که برای W و b های مختلف مشخص می‌کند پیش‌بینی مدل چقدر به مقادیر واقعی نزدیک است.

$$J(w,b) = \frac{1}{2m} \sum_{i=0}^{m-1} \left(f_{w,b}(x^{(i)}) - y^{(i)}\right)^2$$

برای بررسی میزان تفاوت مقادیر پیش‌بینی شده و مقادیر واقعی، از میانگین مربعات تفاوت (Mean Squared Error) آن‌ها به عنوان تابع هزینه استفاده می‌کنیم:

```
def cost_function(y_hat, y):
    return np.mean((y_hat - y)**2) / 2
```

در نتیجه هدف نهایی ما، کاهش مقدار تابع هزینه و پیدا کردن مقدار پارامترهای W و b به گونه‌ای است که $J(w, b)$ را مینیمم کنند. برای این کار، از الگوریتم گرادیان کاهشی استفاده خواهیم کرد.

الگوریتم گرادیان کاهشی (Gradient Descent)

این الگوریتم، روشی را برای بهینه‌سازی تابع ارائه می‌دهد که با تغییر مکرّر و آزمایش W و b های مختلف، مشتقِ جزئیِ تابع هزینه (گرادیان تابع، در ابعاد بزرگ‌تر از 2) را در هر نقطه نسبت به دو پارامتر ذکر شده محاسبه کرده و در خلاف جهت آن حرکت می‌کند تا به نقطه مینیمم (محلی) برسد.

$$\frac{\partial J(w,b)}{\partial b} = \frac{1}{m}\sum_{i=0}^{m-1}(f_{w,b}(x^{(i)}) - y^{(i)})$$

$$\frac{\partial J(w,b)}{\partial w} = \frac{1}{m}\sum_{i=0}^{m-1}(f_{w,b}(x^{(i)}) - y^{(i)})x^{(i)}$$

میزان این حرکت در هر تکرار، به ضریبی به نام نرخ یادگیری (α) بستگی دارد.

$$b = b - \alpha\frac{\partial J(w,b)}{\partial b}$$

$$w = w - \alpha\frac{\partial J(w,b)}{\partial w}$$

شکل کلی تابع گرادیان کاهشی به صورت زیر خواهد بود:

```
def gradient_descent(X, y, W, b, alpha):
    m = X.shape[0]

    error = predict(W, X, b) - y
    dW = X.T @ error / m
    db = np.mean(error)

    W -= alpha * dW
    b -= alpha * db

    return W, b, error
```

در تابع gradient_descent ابتدا تعداد نمونه‌های آزمایشی را به عنوان m تعریف می‌کنیم. تابع predict را فراخوانی می‌کنیم که مقدار پیش‌بینی شده قیمت را با توجه به W و b تعیین شده برمی‌گرداند و مقدار تفاوت آن از مقادیر واقعی را در بردار error ذخیره می‌کنیم. سپس مقدار مشتق جزئی تابع را نسبت به W و b با توجه به فرمول‌های بالا محاسبه کرده و به اندازه α از هر کدام، مشتق‌شان را کم می‌کنیم.

```
def predict(W, X, b):
    return X @ W + b
```

در نهایت، ما این کار را به اندازه‌ای تکرار می‌کنیم که خروجی تابع به یک مقدار خاص همگرا شود. این مقدار مینیمم محلی تابع هزینه ما خواهد بود.

$$repeat\ until\ convergence:\ \{$$
$$b = b - \alpha \frac{\partial J(w,b)}{\partial b}$$
$$w = w - \alpha \frac{\partial J(w,b)}{\partial w}$$
$$\}$$

حالا تابع model که تکرار الگوریتم گرادیان کاهشی و رگرسیون خطی را پیاده‌سازی می‌کند، به صورت زیر خواهد بود:

```
def model(X, y, max_itr = 1000, convergence_threshold = 0.0001):
    start = time.time()

    n = X.shape[1]
    W = np.random.rand(n, 1)
    b = 0
    alpha = 0.01
    cost_history = np.zeros(max_itr)
    itr = 0

    for i in range(max_itr):
        W, b, error = gradient_descent(X, y, W, b, alpha)
        cost_history[i] = np.mean(error**2)
        itr += 1

        if (i + 1) % 20 == 0:
            print(f"Cost Function on Iteration {i + 1} =
{cost_history[i]}")

        if i > 0 and cost_history[i - 1] - cost_history[i] <
convergence_threshold:
            break

    end = time.time()

    return W, b, cost_history, itr, end - start
```

در این تابع، ما مقدار α را 0.01 در نظر گرفته‌ایم. سپس در هر تکرار حلقه (تا زمانی که به مقدار تعیین شده برای حداکثر تکرار برسیم)، الگوریتم گرادیان کاهشی را اجرا کرده و b و W را به‌روز رسانی می‌کنیم و MSE مقدار پیش‌بینی شده و مقدار واقعی را در cost_history مربوط به آن تکرار ذخیره می‌کنیم.

همچنین در کد این قسمت ما محدوده‌ای را برای تکرارها در نظر می‌گیریم. به این صورت که ابتدا الگوریتم تا زمانی تکرار می‌شود که تفاوت مقادیر در دو تکرار متوالی کمتر از 0.0001 شود؛ در این صورت می‌گوییم به یک عدد مشخص همگرا شده‌ایم. در غیر این صورت، اگر تعداد تکرارها بیشتر از max_itr

شود ولی نتیجه به مقداری همگرا نشده باشد (تفاوت اعداد در دو تکرار متوالی کمتر از 0.0001 نشود) اجرای الگوریتم را به پایان می‌رسانیم.

در نهایت مقادیر:

- `W` (بردار ضرایب)،
- `b` (مقدار عرض از مبدا)،
- `cost_history` (بردار هزینه‌های محاسبه شده)،
- `itr` (تعداد تکرار الگوریتم)
- `end - start` (مدت زمان اجرای الگوریتم)

به عنوان خروجی الگوریتم بازگردانده می‌شود.

اجرای برنامه

پس از اجرای کد با مقادیر زیر

```
W, b, cost_history, itr, ex_time = model(X, y, 400)
```

خروجی مدل (چاپ MSE ها در هر 20 تکرار) به شکل زیر خواهد بود:

```
W, b, cost_history, itr, ex_time = model(X, y, 400)

Cost Function on Iteration 20 = 661452719.7945381
Cost Function on Iteration 40 = 457243629.96561486
Cost Function on Iteration 60 = 322169631.66496545
Cost Function on Iteration 80 = 232539508.63447753
Cost Function on Iteration 100 = 172895539.79181936
Cost Function on Iteration 120 = 133103914.30236304
Cost Function on Iteration 140 = 106493704.78467295
Cost Function on Iteration 160 = 88657969.99996744
Cost Function on Iteration 180 = 76676507.45556162
Cost Function on Iteration 200 = 68609167.38252941
Cost Function on Iteration 220 = 63163958.7962194
Cost Function on Iteration 240 = 59478751.91394922
Cost Function on Iteration 260 = 56977189.650518596
Cost Function on Iteration 280 = 55273294.7139131
Cost Function on Iteration 300 = 54108152.97639609
Cost Function on Iteration 320 = 53307794.822564736
Cost Function on Iteration 340 = 52755124.32318504
Cost Function on Iteration 360 = 52371179.829173885
Cost Function on Iteration 380 = 52102605.53854439
Cost Function on Iteration 400 = 51913264.21601968
```

مقادیر به دست آمده برای ضرایب W و b

```
PRICE =
1061.46893392563 * duration +
-1735.2131533324166 * days_left +
-20716.249843495043 * class_Economy +
476.2502135189587 * arrival_time_encode +
-140.37070740776645 * departure_time_encode +
2005.277532176865 * stops_encode +
20532.91607021594
```

نتیجه‌گیری

با استفاده از الگوریتم گرادیان کاهشی و رگرسیون خطی، پس از دو اجرا بر روی دیتاست داده شده (بار اول 20% آموزشی و بار دوم روی 80% آزمایشی)، نتایج زیر را به دست آوردیم:

```
y_pred = predict(W, x_test, b)
```

```
print('R^2:',metrics.r2_score(y_test, y_pred))
print('MAE:',metrics.mean_absolute_error(y_test, y_pred))
print('MSE:',metrics.mean_squared_error(y_test, y_pred))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

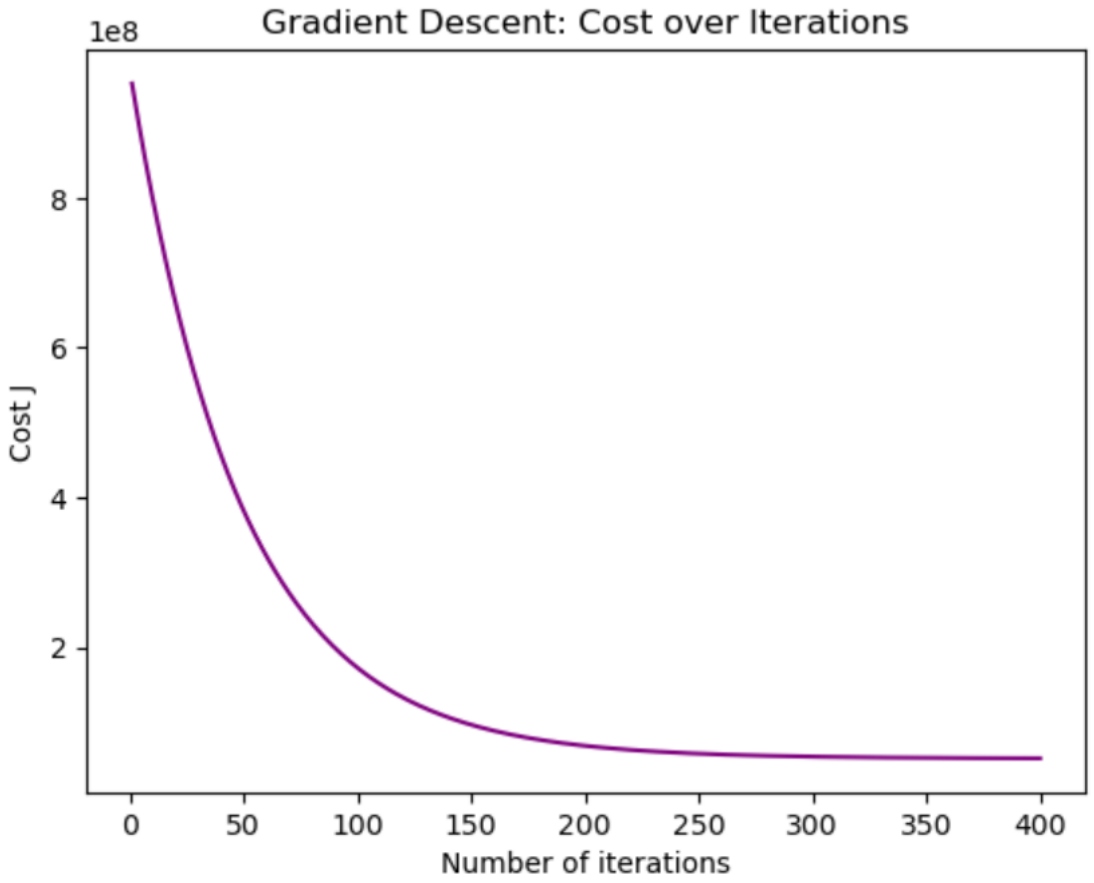
```
R^2: 0.9009362051101376
MAE: 4598.3549258533485
MSE: 50948405.12766634
RMSE: 7137.815150847376
```

```
y_pred = predict(W, X, b)
```

```
print('R^2:',metrics.r2_score(y, y_pred))
print('MAE:',metrics.mean_absolute_error(y, y_pred))
print('MSE:',metrics.mean_squared_error(y, y_pred))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y, y_pred)))
```

```
R^2: 0.8992847714949826
MAE: 4623.778452557727
MSE: 51905398.438425995
RMSE: 7204.540126782971
```

نمودار مقدار تابع هزینه به ازای هر تکرار به صورت زیر می‌باشد:



همچنین در آخرین اجرای الگوریتم، زمان اجرا برابر مقدار زیر بود:

```
print(f"Training Time: = {ex_time}s")
```

```
Training Time: = 1.9172484874725342s
```

کتابخانه‌های مورد استفاده

- pandas
- numpy
- sklearn
- seaborn
- matplotlib
- time

منابع

- Coursera: کورس آقای اندرو ان جی
- W3Schools: رسم نمودارها
- Google.com